In partial fulfillment of the requirements for graduation with the Deans Scholars Honors Degree in Computer Science and Mathematics

# Improving Data Locality of the Nonsymmetric QR Algorithm

By

Kevin Eric Jia

**Supervisors:**

Dr. Robert van de Geijn

Dr. Todd Arbogast

Dr. Enrique S. Quintana-Ortí

# Improving Data Locality of the Nonsymmetric QR Algorithm

By

Kevin Eric Jia

**Department of Computer Science:**

**Printed Name : Kevin Jia**

**Signature : _____ Date : _____**

**Supervisor (Printed): Dr. Robert van de Geijn**

**Signature: _____ Date : _____**

**Supervisor (Printed): Dr. Enrique S. Quintana-Ortí**

**Signature: _____ Date : _____**

# Improving Data Locality of the Nonsymmetric QR Algorithm

By

Kevin Eric Jia

**Department of Mathematics:**

**Printed Name : Kevin Jia**

**Signature : _____ Date : _____**

**Supervisor (Printed): Dr. Todd Arbogast**

**Signature: _____ Date : _____**

# IMPROVING DATA LOCALITY OF THE NONSYMMETRIC QR ALGORITHM

by

Kevin Eric Jia

**Undergraduate Honors Thesis**

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

**Bachelor of Science in Mathematics and Computer Science**

**The University of Texas at Austin**
**December 2013**

# Acknowledgements

I would like to thank Dr. Robert van de Geijn, Dr. Enrique Quintana-Ortí and Dr. Todd Arbogast for their help throughout this process and co-supervising my thesis. I would also like to acknowledge help from Field Van Zee who answered my many questions regarding his optimization of the symmetric QR algorithm.

# Table of Contents

# List of Figures

# Improving Data Locality of the Nonsymmetric QR Algorithm

KEVIN ERIC JIA

## Abstract

The QR algorithm computes the Schur decomposition of a matrix and is the most popular algorithm for solving eigenvalue problems for dense nonsymmetric matrices. The algorithm suffers from a memory access bottleneck though. By restructuring the application of Householder reflectors to the transformation matrix in the nonsymmetric QR algorithm, data locality can be improved, increasing performance. This improvement is demonstrated against the LAPACK implementation of the implicit QR algorithm for nonsymmetric matrices, DLAHQR.

# Chapter 1. Introduction

The QR algorithm computes the Schur decomposition of a square matrix and is the algorithm most widely used for computing all eigenvalues of dense nonsymmetric matrices. The Schur decomposition of an $n \times n$ real-valued matrix $A$ is defined as

$$A = ZTZ^T,$$

where $Z$ and $T$ are both $n \times n$ real-valued matrices, $Z^T Z = I$ and $T$ is an upper quasi-triangular matrix with $1 \times 1$ and $2 \times 2$ blocks corresponding to complex conjugate pairs of eigenvalues of $A$. The matrix $Z$ is referred to as the transformation matrix. The columns of $Z$ form an orthonormal basis for the invariant subspaces of the eigenvalues of $A$. The computation of $Z$ over the eigenvectors is often preferable due to the orthonormal property [5].

The basic QR algorithm computes the Schur form of $A$ by performing a QR decomposition,

$$A = A^{(0)} = QR,$$

where $Q$ is an $n \times n$ orthogonal matrix, and $R$ is an $n \times n$ upper triangular matrix. The factors are multiplied in reverse order, $A^{(1)} = RQ$, and the algorithm iterates using $A^{(1)}$. Each iteration is called a QR step.

The algorithm has seen several improvements since its inception in the early 1960's that have greatly increased performance. Among these is the implicit QR algorithm. The implicit QR algorithm begins by reducing the $n \times n$ real-valued matrix $A$ to upper Hessenberg form[1]. The QR step is then replaced by a Francis step which applies a set of Householder reflectors to the working

[1] An $n \times n$ matrix $H = [a_{ij}]$ is upper Hessenberg if $a_{i,j} = 0$ for all $i > j + 1$.

matrix, performing nearly equivalent calculations to the basic QR algorithm while maintaining and exploiting the upper Hessenberg form of $A$ [7].

Implementations of the QR algorithm suffer from a memory access bottleneck due to level-2 BLAS-like operations from the repeated applications of $O(n^2)$ computation of Householder reflectors on $O(n^2)$ data. These operations limit the ability to attain high performance due to inefficient reuse of data that resides in the cache. The restructured QR algorithm discussed in [9] addresses this issue for the implicit QR algorithm for symmetric matrices. There, Van Zee et al. create a wavefront algorithm that greatly improves the ratio of flops to memory accesses by rearranging the application of Householder reflectors to the transformation matrix. In this thesis we show how the optimizations made in the restructured QR algorithm can be refactored and applied to the nonsymmetric case for the implicit QR algorithm on Hessenberg matrices. We focus on the much simpler implicit QR algorithm so that the problem of applying Householder reflectors is isolated. Modern implementations of the QR algorithm use variations of the multishift QR algorithm and aggressive early deflation, a strategy developed by Braman et al. [1, 2, 3]. Rearranging the application of Householder reflectors for the multishift QR algorithm with aggressive early deflation is left for future research.

The following chapters outline the QR algorithm and investigate the application of a wavefront algorithm to improve data locality in the restructured QR algorithm to the nonsymmetric case in a simplified version of the LAPACK implicit QR routine for nonsymmetric matrices, DLAHQR [5].

# Chapter 2. The QR algorithm

The QR algorithm computes the Schur decomposition of a matrix and can be used to find all the eigenvalues of a matrix. The algorithm has gone through many changes in its practical implementation. This chapter covers the basic QR algorithm, the QR algorithm on upper Hessenberg matrices with shifts and the implicit QR algorithm. Further details on the QR algorithm can be found in [7, 4, 6].

## 2.1. THE BASIC QR ALGORITHM

The basic QR algorithm can be summarized very simply. A QR factorization is performed on an $n \times n$ matrix $A$, reducing it to the product of $Q$, an $n \times n$ orthogonal matrix, and $R$, an $n \times n$ upper triangular matrix. The factors are then multiplied in reverse and the algorithm iterates. The product of the orthogonal matrices are accumulated in $Z$. Algorithm 1 illustrates this process.

**Algorithm 1** The Basic QR Algorithm

$A$ is an $n \times n$ matrix
$A^{(0)} = A$
**for** $(k = 1, 2, \ldots)$ **do**
  $A^{(k-1)} = Q^{(k)} R^{(k)}$ ▷ QR factorization
  $A^{(k)} = R^{(k)} Q^{(k)}$
  $Z^{(k)} = Z^{(k-1)} Q^{(k)}$ ▷ Update transformation matrix
**end for**

Notice that

$$A^{(m)} = (Q^{(m)})^* A^{(m-1)} Q^{(m)}$$

and $Q^{(m)}$ is orthogonal. Therefore, all matrices in the sequence $\{A^{(k)}\}$ are unitarily similar and

have the same eigenvalues. Let $T$ be the real Schur form of $A$. The sequence $\{A^{(k)}\}$ converges to $T$, containing the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ on the diagonal. Assume the QR algorithm computes $T$ on the $k$th iteration. The first column of the product

$$Z^{(k)} = \prod_{n=1}^{k} Q^{(n)} \tag{2.1}$$

is an eigenvector of $A$. For each eigenvector $v$ of $T$, $(Z^{(k)})^* v$ is an eigenvector of $A$. To find the other eigenvectors of $A$, the homogeneous equation

$$(T - \lambda_i)v = 0 \tag{2.2}$$

is solved and $v$ is transformed by $(Z^{(k)})^*$. The basic QR algorithm yields slow convergence rates and is expensive. Each iteration requires computation of the QR factorization of a full matrix, carried out by $\frac{2}{3}n^3$ multiplications and $\frac{2}{3}n^3$ additions, which give a total of $\frac{4}{3}n^3$ flops[1].

## 2.2. THE PRACTICAL QR ALGORITHM

This section covers two major changes to the basic QR algorithm aimed at improving performance. By transforming the initial matrix $A$ to upper Hessenberg form at the beginning the algorithm, the cost of a QR step becomes $O(n^2)$ flops. Convergence of the QR algorithm can be improved by introducing spectral shifts into the algorithm. We will use $H$ to denote Householder reflectors and use $A$ for the working matrix in upper Hessenberg form and $Z$ for the transformation matrix.

[1] A *flop* consists of a single floating point operation, either an addition or a multiplication. The following section improves on both of these issues.

### 2.2.1. Reduction to Upper Hessenberg

Reducing $A$ to upper Hessenberg form helps to reduce high cost of each QR step. The upper Hessenberg form is preserved by each iteration. Define $A^{(0)}$ to be the reduction of $A$ to an $n \times n$ real-valued upper Hessenberg matrix and the transformation matrix $Z^{(0)} = Q$ where $A = QA^{(0)}Q^T$. As

$$Q^{(m)} = A^{(m-1)}(R^{(m)})^{-1}$$

and $(R^{(m)})^{-1}$ is an upper triangular matrix, we see that $Q^{(m)}$ is upper Hessenberg. Then $A^{(m)} = R^{(m)}Q^{(m)}$ must also be upper Hessenberg since the product of an upper Hessenberg matrix and an upper triangular matrix, in any order, is upper Hessenberg.

The matrix can be reduced to upper Hessenberg through the computation and application of a sequence of Householder transforms. These reflectors are applied to the transformation matrix as well. The algorithm has a cost of about $\frac{14}{3}n^3$ flops. This includes forming the transformation matrix. The QR factorization of an upper Hessenberg matrix consists of zeroing out the elements below the diagonal. This means applying $n - 1$ reflectors, each of which costing $O(n)$ flops. Therefore the cost of performing a QR decomposition on an upper Hessenberg matrix is $3n^2$ flops, $O(n)$ faster than performing the basic QR algorithm QR step.

### 2.2.2. Improving the Convergence of the QR Algorithm

The QR algorithm can be accelerated by operating with a shifted matrix $A - \sigma I$ where $\sigma$ is an approximation of an eigenvalue of $A$. Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ denote the eigenvalues of $A$ such that $|\lambda_{i+1}| \geq |\lambda_i|$. The eigenvalues of $A - \sigma I$ are

$$\lambda_1 - \sigma,\ \lambda_2 - \sigma,\ \ldots,\ \lambda_n - \sigma.$$

To show how the use of shifts accelerates the convergence of the QR algorithm, consider $A^{(m)}$ in upper Hessenberg form.

$$A^{(m)} = \begin{bmatrix} a_{11}^{(m)} & a_{12}^{(m)} & \dots & a_{1,n-1}^{(m)} & a_{1,n}^{(m)} \\ a_{21}^{(m)} & a_{22}^{(m)} & \dots & a_{2,n-1}^{(m)} & a_{2,n}^{(m)} \\ & a_{32}^{(m)} & \dots & a_{3,n-1}^{(m)} & a_{3,n}^{(m)} \\ & & \ddots & \vdots & \vdots \\ & & & a_{n,n-1}^{(m)} & a_{n,n}^{(m)} \end{bmatrix} \tag{2.3}$$

As $A^{(m)}$ approaches the triangular form, the diagonal entries approach the eigenvalues of $A$ with $a_{n,n}^{(m)}$ typically approaching the smallest eigenvalue. Therefore we use $a_{nn}^{(m)}$ as an approximation for $\lambda_n$. The subdiagonal entries $a_{i+1,i}^{(m)}$ converge to $0$ linearly with the convergence ratio $\frac{|\lambda_{i+1}|}{|\lambda_i|}$. By shifting the matrix using $\sigma = a_{nn}^{(m)}$, the shifted element $\hat{a}_{n+1,n}^{(m)}$ quickly converges to zero. Once $\hat{a}_{n+1,n}^{(m)}$ is close enough to zero [7], the problem can be deflated so that the next iteration begins with the $(n-1) \times (n-1)$ matrix $\hat{A}^{(m)}$:

$$A^{(m)} + \sigma I = \left[\begin{array}{c|c} \hat{A}^{(m)} & a_{,n}^{(m)} \\ \hline 0 & a_{n,n}^{(m)} \end{array}\right] \tag{2.4}$$

The algorithm then continues with the deflated matrix $\hat{A}^{(m)}$, using the last entry of $\hat{A}^{(m)}$ as the new shift.

A real matrix may have complex eigenvalues. If $\lambda = \alpha + \beta i$ is an eigenvalue of $A$ then its conjugate $\bar{\lambda} = \alpha - \beta i$ is an eigenvalue as well. Collapsing two shifted QR steps using shifts $\sigma_1 = \bar{\sigma}_0$ into a double shift avoids having to work with complex shifts and complex arithmetic.

## 2.3. THE IMPLICIT QR ALGORITHM

The implicit QR algorithm is the version of the QR algorithm used in practice. The algorithm replaces the shifted QR step with the implicitly shifted Francis step, performing almost equivalent computations while increasing the performance and reliability of the practical QR algorithm.

### 2.3.1. The Francis Step

The Francis step carries out the QR step by applying a sequence of Householder reflectors to the upper Hessenberg matrix $A$ rather than subtracting $\sigma I$ and performing a QR factorization. It relies on the implicit $Q$ theorem:

**The Implicit $Q$ Theorem.** Let $A$ be a $n \times n$ real-valued matrix, and assume

$$Q = [q_1, \ldots, q_n] \qquad \text{and} \qquad V = [v_1, \ldots, v_n]$$

are $n \times n$ orthogonal matrices that both similarly transform $A$ to Hessenberg form

$$B = Q^T A Q \qquad \text{and} \qquad C = V^T A V,$$

where $B$ and $C$ are upper Hessenberg. Let $k$ denote the smallest positive integer for which $b_{k+1,k} = 0$, with $k = n$ if $B$ is irreducible.

$(i)$ If $q_1 = v_1$ then

$$q_i = \pm v_i \qquad \text{and} \qquad |b_{i,i-1}| = |c_{i,i-1}| \quad \text{for} \quad i = 2, \ldots, k.$$

$(ii)$ If $k < n$ then $c_{k+1,k} = 0$.

Stated simply, if $B = Q^T AQ$ and $C = V^T AV$, and $Q$ and $V$ have the same first column, then $B$ and $C$ are essentially the same, where $G$ and $H$ may differ only by a trivial similarity transformation. For a proof of the implicit $Q$ theorem, refer to [7, 4].

Each iteration of the Francis step begins by choosing $p$ shifts and computing an $(p+1)\times(p+1)$ Householder reflector $\hat{H}$ that transforms the first $(m+1)$ entries of the first column of $A^{(m)}$ to

$$p(A^{(m)}) = (A^{(m)} - \sigma_1)(A^{(m)} - \sigma_2)\dots(A^{(m)} - \sigma_m)\mathbf{e}_1,$$

where $\mathbf{e}_1 = (1, 0, \dots, 0)^T$ is the unit basis vector. Focus is put on the case $m = 2$ as the LAPACK implementation DLAHQR uses the double shift implicit QR algorithm with the shift strategy discussed in Subsection 2.2.2. $\hat{H}$ is used to form $H$ which applies a similarity transformation on $A$ with the structure

$$H = \left[\begin{array}{c|c} \hat{H} & 0 \\ \hline 0 & I_{n-3} \end{array}\right] = \left[\begin{array}{ccc|c} \times & \times & \times & \\ \times & \times & \times & \\ \times & \times & \times & \\ \hline & & & I_{n-3} \end{array}\right], \tag{2.5}$$

where $I_{n-3}$ is a $(n-3)\times(n-3)$ identity matrix, so that

$$H_1^T AH_1 = \left[\begin{array}{ccc|ccc} \times & \times & \times & \times & \dots & \times & \times \\ \times & \times & \times & \times & \dots & \times & \times \\ \times & \times & \times & \times & \dots & \times & \times \\ \hline \times & \times & \times & a_{44}^{(m)} & \dots & a_{4,n-1}^{(m)} & a_{4,n}^{(m)} \\ & & & a_{54}^{(m)} & \dots & a_{5,n-1}^{(m)} & a_{5,n}^{(m)} \\ & & & & \ddots & \vdots & \vdots \\ & & & & & a_{n,n-1}^{(m)} & a_{n,n}^{(m)} \end{array}\right]. \tag{2.6}$$

The upper Hessenberg form is lost as the $(3,1)$, $(4,1)$, and $(4,2)$ entries are nonzero. This perturbation is the *bulge*. The rest of the Francis step involves the restoration to upper Hessenberg form. This process is usually referred to as *chasing the bulge*. The next Householder reflector $\hat{H}_2$

is chosen to zero out the newly introduced nonzero elements in the first column, resulting in

$$H_2^T H_1^T A H_1 H_2 = \left[\begin{array}{c|ccc|ccc} \times & \times & \times & \times & \dots & \times & \times \\ \hline \times & \times & \times & \times & \dots & \times & \times \\ & \times & \times & \times & \dots & \times & \times \\ & \times & \times & \times & \dots & \times & \times \\ \hline & \times & \times & \times & \dots & a_{5,n-1}^{(m)} & a_{5,n}^{(m)} \\ & & & & \ddots & & \vdots \\ & & & & & a_{n,n-1}^{(m)} & a_{n,n}^{(m)} \end{array}\right] \tag{2.7}$$

The *bulge* has shifted down the diagonal by one position. Next, the third Householder reflector $\hat{H}_3$ restores the upper Hessenberg form in the second column, moving the *bulge* towards the bottom-right corner and this process is continued until the bulge is eventually pushed out of the matrix and upper Hessenberg form is restored. Once chased out we have completed the Francis step. Before the next iteration, the algorithm checks for deflation if any off-diagonal element is close enough to zero.

# Chapter 3. The Restructured QR Algorithm

The restructured QR algorithm presented in [9] stores and applies multiple sets of transformations used in the Francis step of the implicit QR algorithm for symmetric matrices with cache-friendly optimizations that decrease the number of memory accesses. This chapter lightly details the symmetric implicit QR algorithm in order to discuss the wavefront algorithm from [9], and details the application of findings from the restructured QR algorithm to the nonsymmetric implicit QR algorithm for better data locality.

## 3.1. The Restructured QR Algorithm for Tridiagonal Matrices

### 3.1.1. The Symmetric Implicit QR algorithm

Assume $A$ is a real matrix. The implicit QR algorithm is simplified when $A$ is symmetric. The reduction to upper Hessenberg form through unitary transformations on $A$ preserve the symmetric property and results in a tridiagonal matrix:

$$A = \begin{bmatrix} a_{11}^{(m)} & a_{12}^{(m)} & & & \\ a_{21}^{(m)} & \ddots & \ddots & & \\ & \ddots & \ddots & a_{n-1,n}^{(m)} & \\ & & a_{n,n-1}^{(m)} & a_{n,n}^{(m)} & \end{bmatrix} \tag{3.1}$$

Working with a matrix in tridiagonal form allows for compact storage since only the diagonal and off-diagonal need to be stored. Like the nonsymmetric case, the algorithm performs a Francis step and introduces a bulge to the tridiagonal form by applying a transformation. The transformation is a Givens rotation

$$G_0^{(m)} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

that zeroes the first off-diagonal element, $a_{10}$ of the shifted matrix $A^{(m)} - \sigma I$. After the bulge is introduced, it is chased out by applying a sequence of Givens rotations

$$G_1^{(m)}, \quad G_2^{(m)}, \quad \ldots, \quad G_{n-2}^{(m)}$$

and the matrix is returned to tridiagonal form. An important difference between the nonsymmetric and symmetric cases is that the Givens rotations introduce a $2 \times 2$ bulge as opposed to the $3 \times 3$ bulge introduced in the nonsymmetric implicit QR algorithm.

### 3.1.2. The Wavefront Algorithm

Updating the transformation matrix by applying a set of $(n-1)$ Givens rotations one at a time is inefficient and results in performance loss due to an unfavorable ratio of memory operations to computations. For each Francis step, $n^2$ floating point numbers must be loaded from and stored

back to main memory resulting in a theoretical minimum of $2n^2$ memory operations and a ratio of 3 flops per memop[1] when applying the Givens rotations in the traditional ordering. By applying $k$ sets of rotations the algorithm can increase the ratio of flops per memop to $3k$. This can be done by reordering the application of Givens rotations as illustrated in Figure 3.1.2 for $k = 4$.

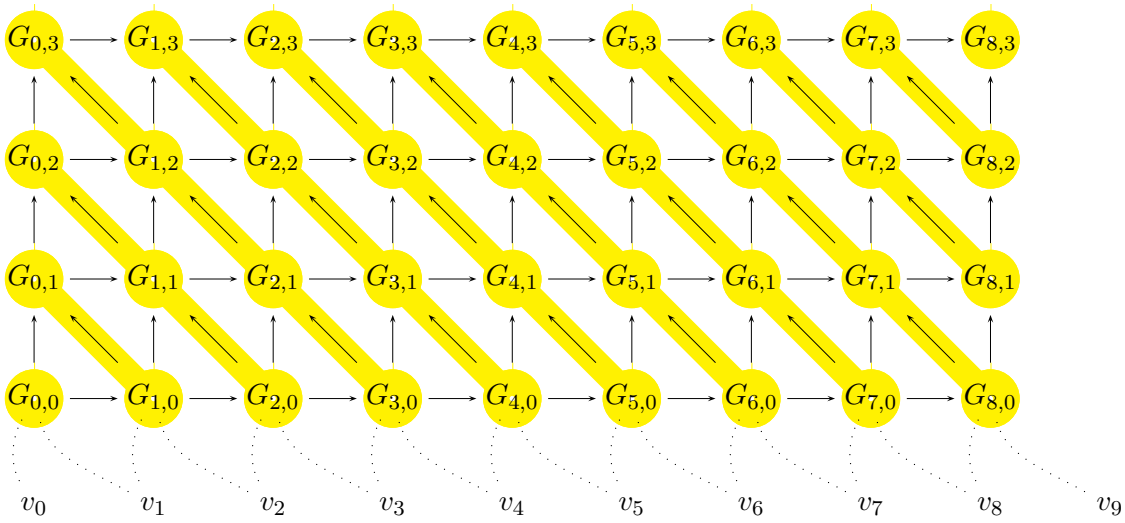


Figure 3.1: A representation of four Francis sets of Givens rotations. $G_{i,j}$ is the $i^{\text{th}}$ Givens rotation of the $j^{\text{th}}$ Francis set, applied to columns $v_i$ and $v_{i+1}$ of $V$. Arrows represent dependencies between Givens rotations. The rotation at an arrow's head is dependent on the rotation at the tail. Givens rotations in the same highlighted region represent a wave. Waves are applied to the transformation matrix left to right and the Givens rotations within a wave are applied from bottom-right to top-left. Figure modeled after figure 4 in [9]

The arrows indicate dependencies between the Givens rotations. The graph is directed-acyclic and therefore defines a partial order. Because of this, if the partial order is obeyed, the rotations are applied to the columns in the same order as the traditional implicit QR algorithm. The dotted lines of the figure represent "waves" of rotations. The waves are applied from left to right and each rotation of a particular wave is applied top to bottom.

By accumulating and reordering the application of the rotations, the wavefront algorithm is able to improve reuse of data in the caches. The ratio of cache misses to total accesses is approximately

[1] A memop will be defined as the reading or writing of a floating-point number.

$\frac{1}{2k}$, and the ratio of cache hits to total accesses is $1 - \frac{1}{2k}$ [9]. For large $k$, the ratio of cache hits to total accesses approaches $1$ implying efficient reuse of data in the cache.

## 3.2. A WAVEFRONT ALGORITHM FOR THE NONSYMMETRIC IMPLICIT QR ALGORITHM

### 3.2.1. Accumulating Householder Reflections

The ideas presented in the wavefront version of the restructured QR algorithm can be adapted for the nonsymmetric case. Let $Z^{(m)}$ be the $n \times n$ transformation matrix at step $m$ of the QR algorithm and $v_i, \ldots, v_n$ be the columns of $Z^{(m)}$. In the LAPACK implementation of the implicit QR algorithm, DLAHQR, transformations are applied in an interleaving pattern first to the $n \times n$ matrix $A^{(m)}$ and then to the $n \times n$ transformation matrix $Z^{(m)}$. Observe that by simply accumulating the Householder reflectors of a Francis step, and applying them at once to $Z$, the algorithm may benefit from better cache reuse. Application of a single Householder reflector in the LAPACK routine needs cache space for columns of $Z^{(m)}$ and $A^{(m)}$ before any cache reuse. If the reflectors are applied in succession, the cache sees reuse with around $2n$ space when updating the transformation matrix.

### 3.2.2. Restructuring the Francis Step

Accumulating the Householder reflectors provides a small improvement in performance and data locality. Data localization can be further improved through a reordered application of Householder reflectors. To do this, $k$ Francis sets are accumulated. We use the notation $H_{i,m}$ to represent $i$th Householder reflector from the $m$th set. Each application of a Householder reflector $H_{i,m}$

affects three columns of the transformation matrix $Z^{(m)}$, specifically

$$z_i^{(m)}, \quad z_{i+1}^{(m)}, \quad z_{i+2}^{(m)}.$$

The graph displayed in Figure 3.2 represents the dependencies between four Francis sets of Householder reflectors for the implicit QR algorithm. Similar to Figure 3.1, the highlighted sections represent waves and the arrows indicate dependencies between Householder reflectors. This creates a partial ordering of the Householder reflectors. A workspace size $3k(n-1)$ is needed to store $k$ Francis sets of Householder reflectors. If the waves are applied left to right and the elements of the waves applied bottom-right to top-left, the partial ordering is preserved with respect to the application order of the traditional implicit QR algorithm. This means the reflectors are applied to columns of $Z$ in the exact same order as the traditional implicit QR algorithm.

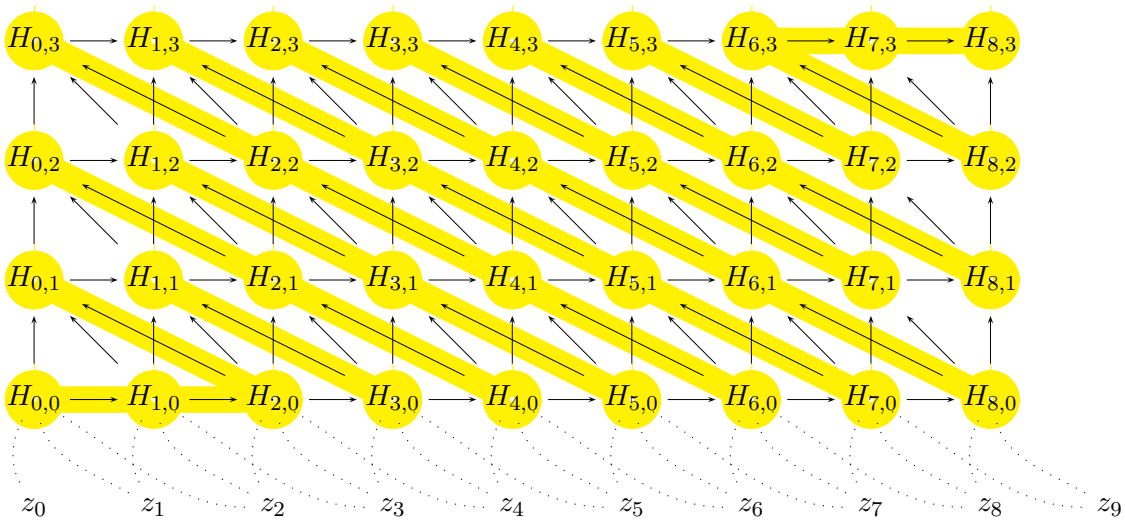


Figure 3.2: A representation of four Francis sets of Householder reflectors. $H_{i,j}$ is the $i^{\text{th}}$ Householder reflector of the $j^{\text{th}}$ Francis set which is applied to columns $v_i$, $v_{i+1}$, $v_{i+2}$ of $Z$. Arrows represent dependencies between Householder reflectors. The reflector at an arrow's head is dependent on the reflector at the tail. Householder reflectors in the same highlighted region represent a wave. Waves are applied to the transformation matrix left to right and the Householder reflectors within a wave are applied from bottom-right to top-left.

For each wave, only one new column of $Z^{(m)}$ is loaded into memory. The implicit QR algorithm with accumulation of Householder reflectors loads one new column into cache for each

Householder reflector applied. The number of sets accumulated, $k$, is limited by the size of the cache. Assume the cache is fully-associative and uses the least-recently used replacement policy. Assume $n_c$, the size of the cache, is significantly smaller than $n^2$ but larger than $kn$ where $k \ll n$. We will also simplify our analysis by counting column accesses, since every application of a reflector touches an entire column of data. The traditional QR algorithm accesses most of the $n$ columns of $Z$ three times for a total of $3n - 4$ column accesses. Of those accesses, $n$ are cache misses and $2(n - 2)$ are cache hits. Since relevant data has likely been evicted by the beginning of the next Francis step, for $k$ steps, there are $kn$ cache misses and $2k(n - 2)$ cache hits, resulting in a hit rate of approximately $67\%$ for large enough $n$.

Using the refactored wavefront algorithm, there is a cache miss for each wave applied excluding the first, resulting in $n + 1$ cache misses. Since there are $k(3n - 4)$ accesses, the ratio of cache misses to total accesses is

$$\frac{n+1}{k(3n-4)} \approx \frac{1}{3k}$$

making the fraction of cache hits $1 - \frac{1}{3k}$. For large $k$, this ratio approaches $1$, exhibiting efficient data accesses.

Algorithm 2 illustrates the implementation of the refactored wavefront algorithm. ApplyHSingle applies the Householder reflectors to $Z$ in the traditional ordering.

**Algorithm 2** $[Z] :=$ APPLYWAVE$(k, m, n, T, Z)$

**if** $(n < r$ Or $k = 1)$ **then**
  $Z :=$ APPLYHSINGLE$(k, m, n, T, Z)$ **return**
**end if**
**for** $(j := 0; j < 2(k-1); ++j)$ **do** ▷ Startup phase
  **for** $(i := m, g := 1; i > 1; i -= 2, ++g)$ **do**
    $Z = H_j^{(g)} Z$ ▷ Apply Householder
  **end for**
**end for**
**for** $(j := 2(k-1)+1; j < n-1; ++j)$ **do** ▷ Pipeline phase
  **for** $(i := 0; i < k; ++i$ **do**
    $Z = H_j^{(i)} Z$
  **end for**
**end for**
**for** $(j := n - 2(k-1); j < n-1; ++j)$ **do** ▷ Shutdown phase
  **for** $(i := 2 + cnt/2, g := j + 2(k-i), cnt = 0; i < k; ++i, ++cnt)$ **do**
    $Z = H_i^{(g)} Z$ ▷ Apply Householder
  **end for**
**end for**

# Chapter 4. Performance

In this chapter we discuss details of our implementation and experiments. Three different versions of the implicit QR algorithm are used for testing including our implementation of the wavefront algorithm. Each of these versions are covered in detail. Finally, results are shown, and the performance of the three versions are analyzed.

## 4.1. Platform and Implementation Details

Experiments were performed on a single core of a Intel Xeon X7460 2.66GHz on a Dell PowerEdge R900 server. Each core possesses a 1 megabyte L2 cache with a peak performance of 10.64 Giga FLoating-point Operations Per Second (GFLOPS). Experiments were run under the Linux Server 2.6.18–348 operating system. The experiments were run on upper Hessenberg matrices with double precision floating-point arithmetic generated by running the LAPACK routine for reducing a matrix to upper Hessenberg form, DGEHRD, on matrices with randomly generated elements. The code was compiled in gcc version 4.1.2 with optimization flag -03.

Three implementations of the QR algorithm were tested. The first version is a slightly modified implementation of DLAHQR of the LAPACK library version 3.4.2. DLAHQR computes the Schur factorization of an upper Hessenberg matrix using the implicit double shift QR algorithm and the matrix of Schur vectors. Modifications were made so that the algorithm is only able to deflate when the last one or two subdiagonal elements are close enough to zero. In the original implementation, if a small enough subdiagonal element is found, the matrix is split off and the eigenvalues and Schur

form of the smaller matrix are computed first. This modification was made to further simplify the LAPACK implementation. By limiting deflation, we are able to focus on the performance increase given by our implementation of the wavefront algorithm. This version of the algorithm uses the interleaved application of Householder reflectors described in Subsection 3.2.1.

The second version is a further modification of version one and the DLAHQR routine. This implementation focuses on the accumulation of Householder reflectors covered in Subsection 3.2.1. The implementation accumulates $k$ Francis sets before applying any Householder reflectors, removing the interleaving pattern. A workspace of size $3k(n-1)+2k$ is required to store $k$ Francis sets of reflectors and the dimensions of the active matrix at the end of each implicit QR step.

The third version adds the refactored wavefront algorithm shown in Algorithm 2 to the second version. We accumulate $k$ Francis sets and then apply them to the transformation matrix in an ordering outlined in Subsection 3.2.2. The algorithm requires $3k(n-1)+2k$ workspace.

## 4.2. RESULTS

Figure 4.1 shows performance of each version of the implicit QR algorithm, computing the full set of eigenvalues and eigenvectors. Versions 2 and 3 of the algorithm are run for $k = 4, 16, 32$ Francis sets accumulated before applying Householder reflectors. For small matrices, versions 1 and 2 perform similarly, due to the cache being large enough to hold the entire transformation matrix and $A^{(m)}$. Version 3 loses performance because of high overhead calculations for the refactored wavefront algorithm. For large matrices, version 3 outperforms all other implementations. For large matrices, the implementation with the refactored wavefront algorithm outperforms the slightly modified DLAHQR implementation by about $30\%$.

Figure 4.2 shows the performance of updating the transformation matrix when computing the Schur form. This is obtained by running each version of the algorithm on an upper Hessenberg matrix $A$ twice. The first run computes the Schur form $T$ of $A$ and updates the transformation matrix and the second computes the Schur form without updating the transformation matrix. The difference in run times is used to compute the GFLOPs of each version of the algorithm. The wavefront version maintains stable performance for large matrices, while the other two versions rapidly decrease in performance around $n = 400$. This is due to the transformation matrix no longer fitting in the L2 cache. At $n = 400$, the transformation matrix is $160000$ double precision numbers, each taking $8$ bytes. This is about 1 megabyte, the size of the L2 cache. The results show that the wavefront version of the implicit QR algorithm improves performance by more than a factor of 2 when compared to the DLAHQR implementation.

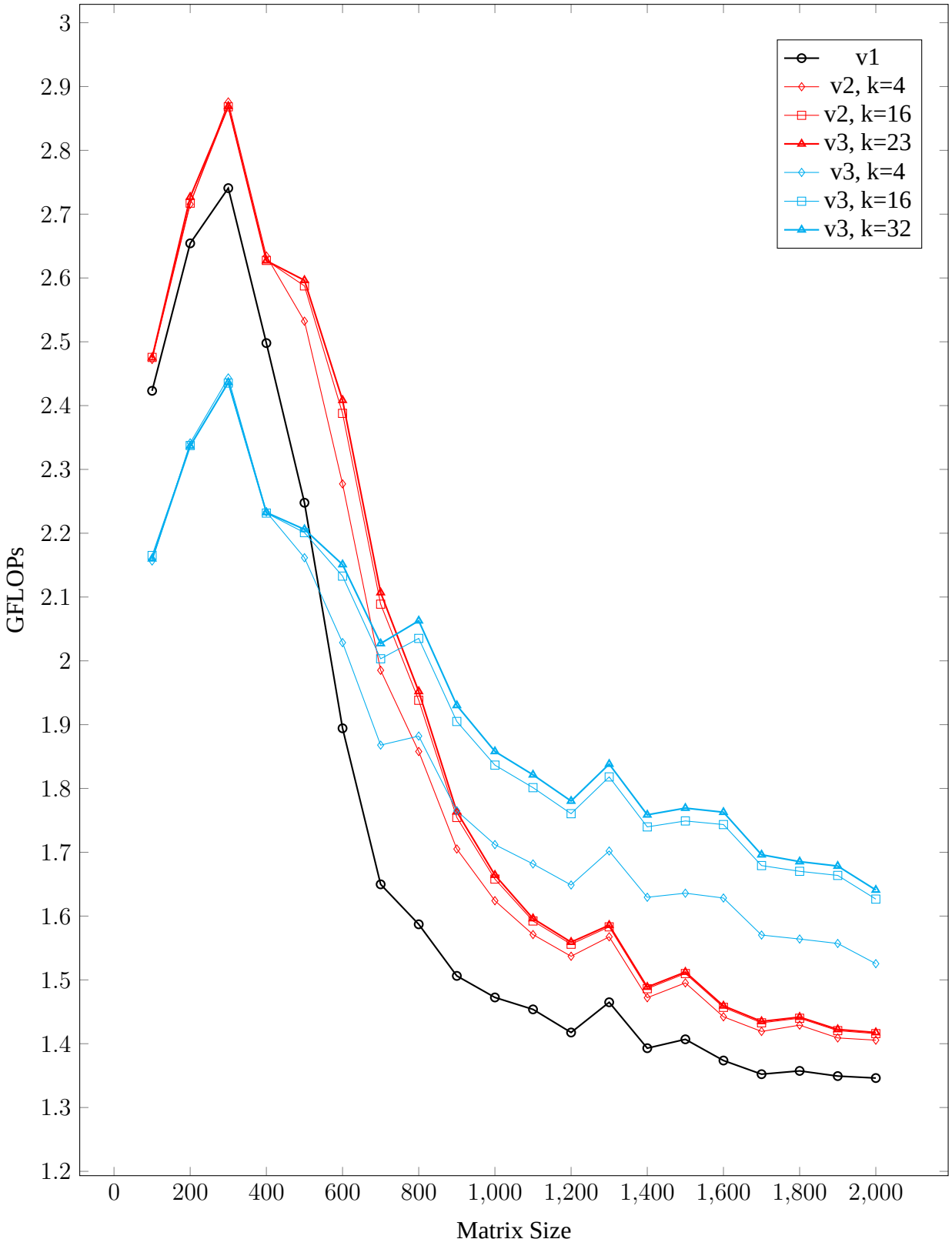


Figure 4.1: Performance of three versions of implicit QR algorithm with Schur form computation. $25n^3$ flops is used as an estimation of the upcount for all three versions. $k$ represents the accumulation factor of Francis sets for versions 2 and 3.

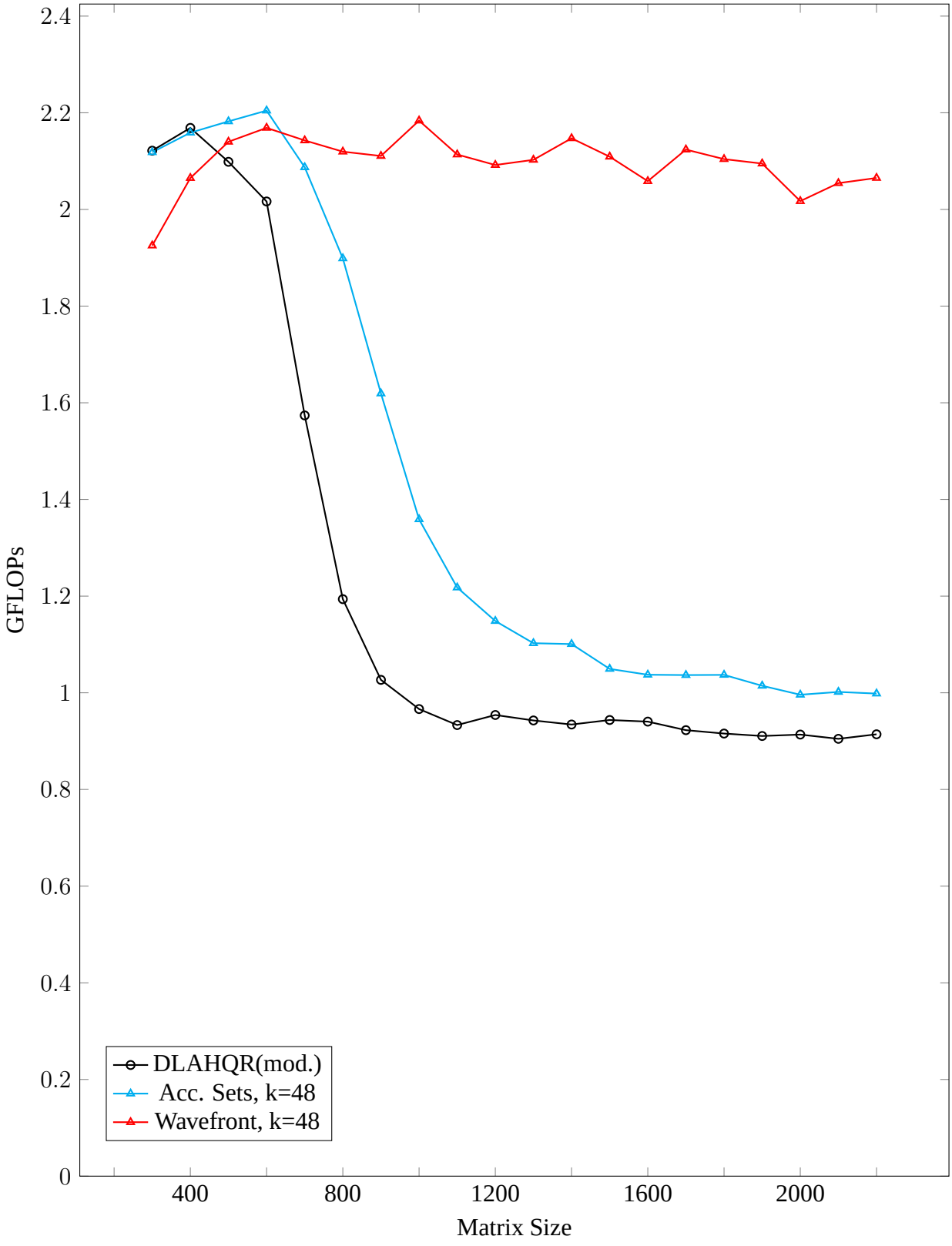


Figure 4.2: Performance of transformation matrix update of three versions of implicit QR algorithm with Schur form computation. $k$ represents the accumulation factor of Francis sets for versions 2 and 3.

# Chapter 5. Conclusion

The QR algorithm is a beautifully simple algorithm that computes the Schur decomposition of a matrix. After over 35 years, it is still the most popular approach for stably solving eigenvalues of nonsymmetric dense matrices. The basic QR algorithm and practical QR algorithm are covered. The wavefront algorithm used in the restructured QR algorithm is then discussed. By applying data localization shown in the restructured QR algorithm for symmetric matrices, performance can be increased for the implicit QR algorithm for nonsymmetric matrices. The results show that cache reuse afforded by reordering and accumulating the application of Householder reflectors improves performance for large matrices. Further work can be done by applying the principles shown in this paper to modern implementations of the QR algorithm.

# Bibliography


[1] Z. Bai, J. Demmel, *On a block implementation of Hessenberg QR iteration*, Intl. J. of High Speed Comput., 1, pp. 97112, 1989.

[2] K. Braman, R. Byers, R. Mathias, *The Multishift QR Algorithm. Part I: Maintaining Well-focused Shifts and Level 3 Performance*, SIAM J. Matrix Anal. Appl. 23, 929-947.

[3] K. Braman, R. Byers, R. Mathias, *The Multishift QR Algorithm. Part II: Aggressive Early Deflation*, SIAM J. Matrix Anal. Appl. 23, 948-973.

[4] G. H. Golub, C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, 1996.

[5] E. Anderson, Z. Bai, C, Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorenson, *LAPACK Users' Guide*, Third Edition, Society for Industrial and Applied Mathematics, 1999, Philadelphia, PA, 0-89871-447-8.

[6] Peter Arbenz, *Numerical Methods for Solving Large Scale Eigenvalue Problems*, http://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter3.pdf.

[7] David S. Watkins, *Fundamentals of Matrix Computations*, John Wiley & Sons, New York, 1991.

[8] David S. Watkins, *The QR Algorithm Revisited*, SIAM Review, Vol. 50, pp. 133-145, 2008.

[9] Field G. Van Zee, Robert van de Geijn, and Gregorio Quintani-Orti, *Restructuring the QR algorithm for High-Performance Application of Givens Rotations*, Technical Report TR-11-36, The University of Texas at Austin, Department of COmputer Science, 2011. FLAME Working Note # 60.